Programación

Aplicaciones Gráficas

SWING

Unidad 13

Jesús Alberto Martínez
versión 0.2

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante	 Atención	 Interesante

Unidad 13. Aplicaciones Gráficas SWING

1 Introducción

Java permite realizar aplicaciones de escritorio en entorno gráfico. Al contrario que en los programas que hemos desarrollado hasta ahora, los programas con interfaz gráfica seguirán un flujo guiado por los eventos que ocurran, generalmente ocasionados voluntariamente con las acciones del usuario, por ejemplo, cuando se pulse un botón. Para ello utilizaremos Swing, que es una herramienta de interfaz gráfica de usuario (GUI) ligera que incluye un amplio conjunto de widgets (botones, cuadros de texto, menús, etc) para aplicaciones Java, y es independiente de la plataforma.

Podemos crear nuestra aplicación gráfica mediante código, pero es más cómodo crearlo a partir de las utilidades de un IDE como Netbeans, Eclipse o IntelliJ. Usaremos ambos métodos, pero, cuando creemos el interfaz gráfico desde el IDE, comprobaremos el código que genera.

2 Programación guiada por Eventos

2.1 ¿Qué es un evento?

Es todo hecho que ocurre mientras se ejecuta la aplicación. Normalmente, llamamos evento a cualquier interacción que realiza el usuario con la aplicación, como puede ser:

- Pulsar un botón con el ratón,
- Hacer doble clic,
- Pulsar y arrastrar,
- Pulsar una combinación de teclas en el teclado,
- Pasar el ratón por encima de un componente,
- Salir el puntero de ratón de un componente,
- Abrir una ventana, etc.

2.2 ¿Qué es la programación guiada por eventos?

Imagina la ventana de cualquier aplicación, por ejemplo, la de un procesador de textos. En esa ventana aparecen multitud de elementos gráficos interactivos, de forma que no es posible que el programador haya previsto todas las posibles entradas que se pueden producir por parte del usuario en cada momento.

Con el control de flujo de programa de la **programación imperativa**, el programador tendría que estar continuamente leyendo las entradas (de teclado, o ratón, etc) y comprobar para cada entrada o interacción producida por el usuario, de cual se trata de entre todas las posibles, usando estructuras de flujo condicional para ejecutar el código conveniente en cada caso. Para cada opción del menú, para cada botón o etiqueta, para cada lista desplegable, y por tanto para cada componente de la ventana, incluyendo la propia ventana, habría que comprobar todos y cada uno de los eventos posibles, por lo que las posibilidades son casi infinitas, y desde luego impredecibles. Por tanto, de ese modo es imposible solucionar el problema.

Para abordar el problema de tratar correctamente las interacciones del usuario con la interfaz gráfica de la aplicación hay que cambiar de estrategia, y la **programación guiada por eventos** es una buena solución.

El proceso sería algo así: cada vez que el usuario realiza una determinada acción sobre una aplicación que estamos programando en Java: un clic sobre el ratón, presionar una tecla, etc, se produce un evento que el sistema operativo transmite a Java, creando un objeto de una determinada clase de evento, y este evento se transmite a un determinado método para que lo gestione. Ejemplos de fuentes de eventos pueden ser:

- Botón sobre el que se pulsa o pincha con el ratón.
- Campo de texto que pierde el foco.
- Campo de texto sobre el que se presiona una tecla.
- Ventana que se cierra.
- Etc.

Un ejemplo típico será la selección de un elemento, bien haciendo click con el ratón sobre él o pulsando [Enter] cuando tiene el foco.

3 Creando Una Aplicación Gráfica

Antes de ver el detalle de todos los elementos que podemos incluir en nuestra aplicación gráfica y ver los eventos que se pueden producir, vamos a ver como crearíamos una aplicación gráfica sencilla, primero utilizando un editor visual y luego codificando los componentes desde cero.

3.1 Aplicación Gráfica codificada desde cero

Vamos a ver como se crearía una aplicación gráfica con Swing sin ningún asistente, programada “a mano”.

1.- Primero crearíamos una clase de tipo ventana (hija de JFrame) que contendrá distintos elementos sobre los que luego programaremos acciones en los diferentes eventos.

```
import java.awt.EventQueue;
import javax.swing.*;
import java.awt.event.*;
public class EjemploJFrame extends JFrame implements ActionListener {
}
```

2.- Dentro de la clase declararíamos las variables de los elementos presentes en la ventana:

```
private JLabel etiq;
private JTextField campoTexto;
private JButton btnPulsame;
private JPanel PanelContenido;
```

Y el main() que lanza la aplicación, en una versión sencilla:

```
public static void main(String[] args) {
   EjemploJFrame frame = new EjemploJFrame();
   frame.setVisible(true)
}
```

O bien de forma más sofisticada:

```
public static void main(String[] args) {
   EventQueue.invokeLater(new Runnable() {
         public void run() {
               try {
         EjemploJFrame frame = new EjemploJFrame();frame.setVisible(true);
               } catch (Exception e) {
                     e.printStackTrace();
               }
         }
   });
}
```

3.- Dentro de la clase EJemploJFrame, con el constructor de la ventana, en la que se definen ciertas características de la ventana, se definen también los componentes de la misma (etiquetas, botones, etc.) y se añaden a la ventana.

Realmente los elementos se añaden a un Panel, que es un contenedor, un elemento intermedio entre ellos y el JFrame.

```
public EjemploJFrame() {
    setTitle("Titulo de la ventana");        // Título opcional
    setBounds(400, 200, 655, 520);           // Cordenadas 'x' 'y',altura,long
    setDefaultCloseOperation(EXIT_ON_CLOSE);// Al cerrar la ventana finaliza
    setVisible(true);                        // La mostramos
```

```
    PanelContenido = new Jpanel();          //Creamos el panel
    PanelContenido.setLayout(null);         //Indicamos su diseño
    setLocationRelativeTo(null);            //Centrada en pantalla
    setContentPane(PanelContenido);         //asigno el pannel a la ventana

    //Componentes
    etiq = new JLabel("Texto de la etiqueta);
    etiq.setBounds(369, 32, 229, 25);
    PanelContenido.add(etiq);
    campoTexto = new JTextField();
    campoTexto.setBounds(371, 68, 193, 26);
    PanelContenido.add(campoTexto);
    JButton btnPulsame = new JButton("Pulsame");
    btnPulsame.setBounds(43, 70, 89, 23);
    PanelContenido.add(btnPulsame);
    //Ahora definiríamos los eventos y sus acciones asociadas
} //fin constructor
```

3.2 Aplicación Gráfica con un asistente

Se explicará en clase el uso adecuado del asistente del entorno de desarrollo que se esté usando.

4 Componentes Swing

Swing ofrece dos tipos de elementos clave: componente y contenedor. Sin embargo, esta distinción es principalmente conceptual debido a que todos los contenedores también son jerárquicamente componentes. La diferencia entre los dos se encuentra en su propósito: como el término se emplea comúnmente, un componente es un control visual independiente, como un botón o un deslizador. Un contenedor aúna a un grupo de componentes. Por ello, un contenedor es un tipo especial de componente que está diseñado para poseer a otros componentes.

Además, para que un componente sea desplegado debe ser colocado en un contenedor. Por ello, todas las interfaces gráficas hechas con Swing contienen al menos un contenedor por defecto. Debido a que los contenedores son componentes, un contenedor puede también poseer a otros contenedores. Esto le permite a Swing definir lo que se denomina una contención jerárquica, en cuyo nivel más alto se encuentra lo que se denomina un contenedor raíz.

4.1 Componentes

En general los componentes de Swing se derivan de la clase JComponent. Las únicas excepciones a esto son los cuatro contenedores raíz que se describen en la siguiente sección. Todos los componentes de Swing están representados por clases definidas en el paquete javax.swing. Observe que todas las clases

comienzan con la letra J. Por ejemplo, la clase para crear una etiqueta es Jlabel; la clase para un botón es JButton; y la clase para un scrollbar es JScrollBar.

4.2 Contenedores

Swing define dos tipos de contenedores. El primero son los contenedores raíz: JFrame, Japplet, JWindow y JDialog. A diferencia de los otros componentes de Swing, los contenedores raíz son componentes pesados y son los que interactúan con el sistema operativo. Como el nombre lo indica un contenedor raíz debe estar en lo alto de una jerarquía de contención y non está contenido ningún otro componente. Además, todas las jerarquías de contención deben comenzar con un contenedor raíz. El contenedor utilizado más comúnmente en aplicaciones es JFrame. El contenedor utilizado para applets es JApplet.

El segundo tipo de contenedores soportados por Swing son los contenedores ligeros. Los contenedores ligeros heredan de la clase JComponent. Por ejemplo, JPanel, el cual es un contenedor de propósito general. Los contenedores ligeros se utilizan a menudo para organizar y administrar grupos de componentes relacionados debido a que un contenedor ligero puede ser contenido por otros contenedores. Así, el programador puede utilizar contenedores ligeros como JPanel para crear subgrupos de controles relacionados que están contenidos en un contenedor exterior.

4.3 Ventana de Aplicación

JFrame: Es el contenedor de alto nivel más empleado, tiene las funcionalidades típicas de una ventana (maximizar, cerrar, título, etc.). Algunos métodos importantes son:

- setSize (w,h); asigna las dimensiones de la ventana (ancho y alto).
- setTitle (str), getTitle ();
- setVisible (bool);
- setDefaultCloseOperation (constante); controla la acción al pulsar la "X" de cierre de ventana. Su valor habitual suele ser JFrame.EXIT_ON_CLOSE.
- setLocationRelativeTo(null); centra la ventana en la pantalla.

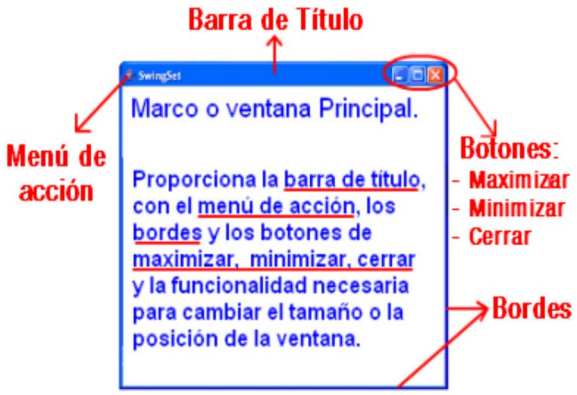
Una vez creado el objeto de ventana, hay que establecer su tamaño, establecer la acción de cierre y hacerla visible.

```
import javax.swing.*;
public class VentanaTest {
    public static void main(String[] args) {
        JFrame f = new JFrame("Titulo de ventana");
        f.setSize(400, 300);
        f..setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        setVisible(true);
    }
}
```

Acciones de cierre:

- JFrame.EXIT_ON_CLOSE: Abandona aplicación.
- JFrame.DISPOSE_ON_CLOSE: Libera los recursos asociados a la ventana.
- JFrame.DO_NOTHING_ON_CLOSE: No hace nada.
- JFrame.HIDE_ON_CLOSE: Cierra la ventana, sin liberar sus recursos

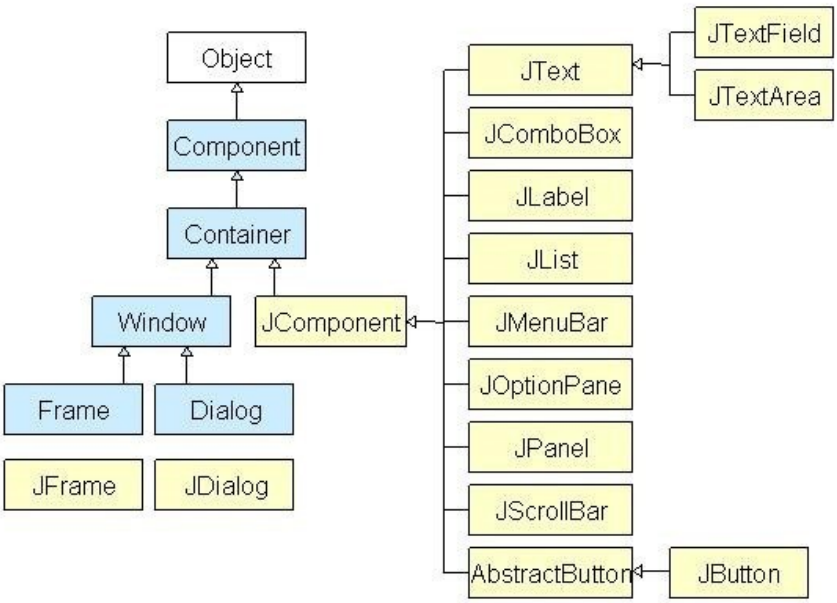


Por último, para cerrar una ventana liberando todos sus recursos, emplearemos el método dispose(). Si tenemos un botón de salir en nuestra ventana, su código podría ser: this.dispose().

4.4 Componentes básicos

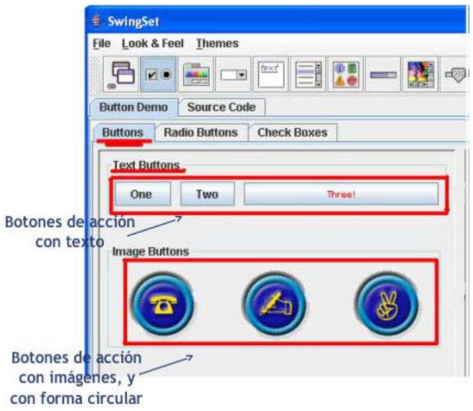
Vamos a introducir aquí los componentes habituales de una ventana de aplicación, con su descripción, apariencia y métodos interesantes. Estos componentes se añadirán a la ventana o, mejor dicho, a un panel de la ventana. Si no hay ningún panel, se añaden al panel por defecto.

Esta es la jerarquía de componentes de Swing.



JButton

Botón de acción. Para realizar alguna acción o proceso.



El texto que muestra se establece con el método setText().

Antes vimos que ActionPerformed era el evento típio de seleccionar/pulsar el botón.

```
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        // Código que se ejecuta al pulsar el botón
    }
});
```

Si deseamos detectar el evento de clicar con el botón derecho, este sería el evento:

```
jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        TareaAlPulsarRaton(evt);
    }
});
private void TareaAlPulsarRaton(java.awt.event.MouseEvent evt) {
    if (((java.awt.event.MouseEvent) evt).getButton() ==
        java.awt.event.MouseEvent.BUTTON3) {
        // Aquí el código:
    }
}
```

JCheckBox

Casilla de verificación. Se usa normalmente para indicar opciones que son independientes, pudiéndose seleccionar algunas, todas o ninguna

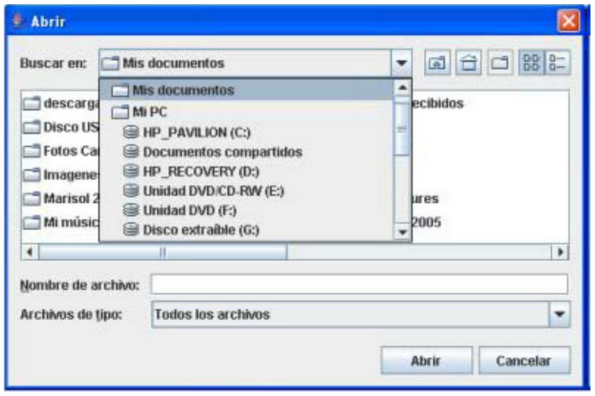
El método que más nos interesa de esta clase es: isSelected() que devuelve true si el check-box está seleccionado y false en caso contrario.

JColorChooser

Panel de controles que permite al usuario seleccionar un color.

JFileChooser

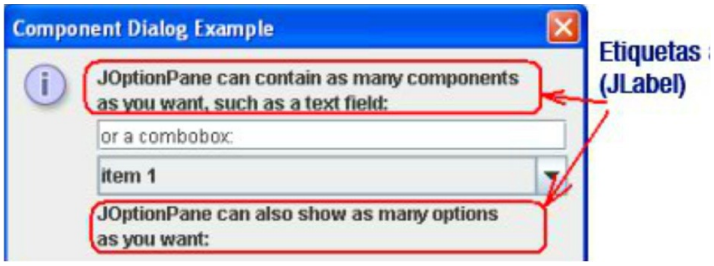
Permite al usuario elegir un archivo para la lectura o escritura, mostrando una ventana que nos permite navegar por los distintos discos y carpetas de nuestro ordenador



Jlabel

Una etiqueta que puede contener un texto corto, o una imagen, o ambas cosas. En general suelen ser estáticas, de forma que desde el asistente de Netbeans le asignaremos un valor y ya no lo modificaremos

más, pero siempre podremos emplear los métodos setText() y getText() para modificar y obtener el texto mostrado, respectivamente.

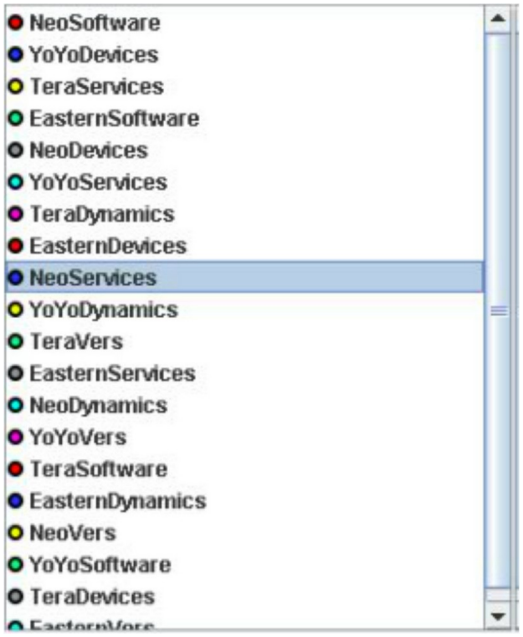



```
jLabel1.setText("Temperatura Celsius");
```

> 📣 Se suele utilizar esta etiqueta para incorporar imágenes.

Jlist

Un cuadro de lista para seleccionar uno o varios elementos de una lista



Los valores contenidos en la lista se gestionan desde una colección (similar a un array) llamada model y que dispone de métodos para añadir / modificar / eliminar elementos y otras operaciones.

Por ejemplo, podemos definir en nuestro frame, como atributo global:

```
DefaultListModel lista1modelo;
```

Al inicializar los componentes:

```
lista1modelo = new DefaultListModel();
jList1.setModel (lista1modelo);
```

y luego podemos emplear los métodos siguientes para trabajar con la lista:

```
lista1modelo.addElement(str);  //añadir a la lista, generalmente String
lista1modelo.clear();
lista1modelo.removeElementAt(position);
lista1modelo.setElementAt(string, position);
lista1modelo.getElementAt(posistion);
lista1modelo.size();
```

Para obtener el índice o valor de la lista seleccionado por el usuario, haremos respectivamente:

```
int index = jList1.getSelectedIndex();
String s = (String) jList1.getSelectedValue();
```

Si no hay ningún elemento seleccionado, getSelecteIndex devuelve -1.

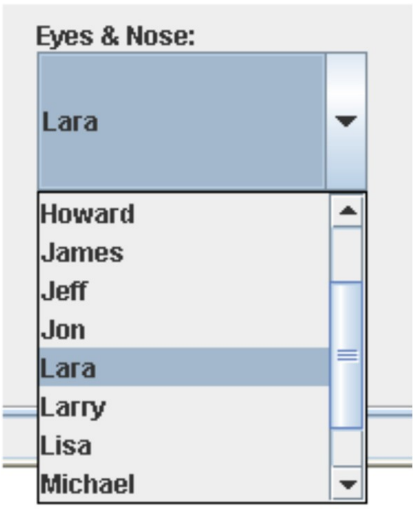
En cuanto a los eventos, nos interesa detectar cuando el usuario selecciona un valor, esto lo hacemos con jListValueChanged (). Por ejemplo:

```
private void jList1ValueChanged(javax.swing.event.ListSelectionEvent evt) {
   String s = (String) jList1.getSelectedValue();
}
```

JcomboBox

Lista desplegable con características similar a JList, solo cambia su apariencia. El model en este caso es de tipo: DefaultComboBoxModel.

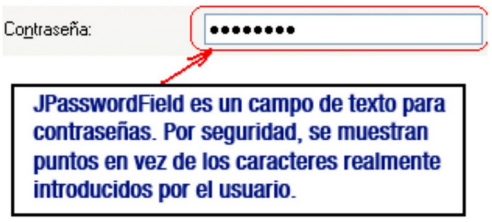
```
lista1modelo = new DefaultComboBoxModel();
jComboBox1.setModel (lista1modelo);
lista1modelo.addElement(...)
```



DefaultComboBoxModel tiene otro constructor al que se le puede pasar como parámetro un array de elementos. En ese caso inicializa la combo con dichos elementos, evitándonos tener que hacer addElement para cada uno.

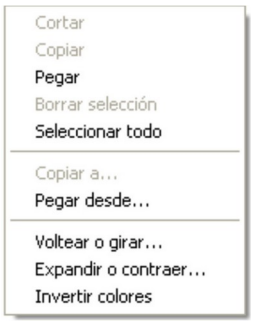
JPasswordField

Un cuadro de texto donde no se muestran los caracteres que realmente se introducen, sino algún otro (puntos o asteriscos). Se usa para introducir contraseñas de forma que otras personas no puedan ver el valor introducido



JPopUpMenu

Un menú emergente. La imagen del ejemplo está sacada de una aplicación de dibujo



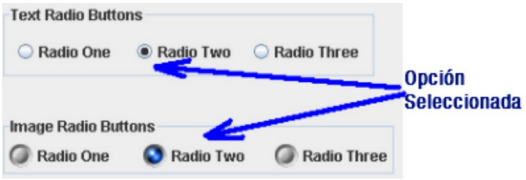
JProgressBar

Una barra de progreso, que visualiza gráficamente un valor entero en un intervalo

Barra de progreso (JProgressBar). Se asocia a una tarea para mostrar de una forma gráfica la parte de tarea que ya se ha realizado y la que queda pendiente. Al mismo tiempo sirve para que el usuario compruebe que la tarea progesa, que el ordenador no se ha quedado bloqueado. En este ejemplo se asocia a la carga del texto en el editor.

JRadioButton

Crea un botón de radio. Se usa normalmente para seleccionar una opción de entre varias excluyentes entre sí



Los botones de radio deben estar agrupados para que, cuando se seleccione uno, el resto del grupo estén seleccionados (es lo que lo diferencia de un check buttonI). Para ello, necesitamos un nuevo componente ButtonGroup, al que añadiremos los botones mediante el método add().

```
ButtonGroup bg=new ButtonGroup();
bg.add (radiobutton1);
bg.add (radiobutton2);
bg.add (radiobutton3);
```

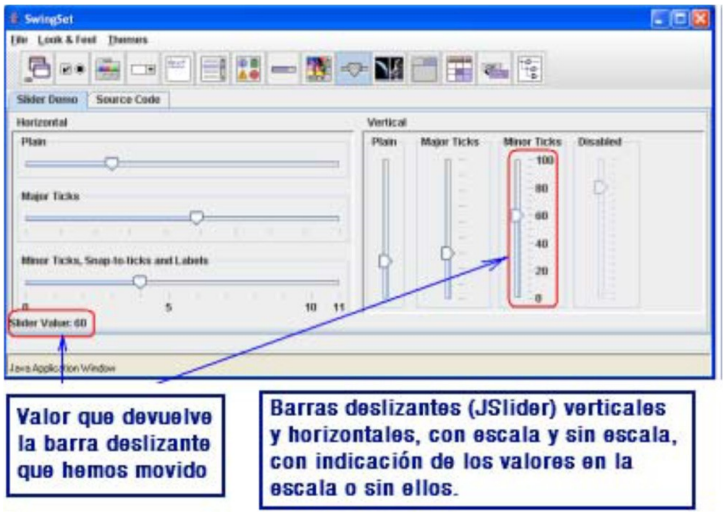
Como en el caso de los check-buttons, el método isSelected() nos informará si un botón está seleccionado o no.

```
if (!jRadioButton1.isSelected() && !jRadioButton2.isSelected()) {
   JOptionPane.showMessageDialog(null,"Seleccione una opción");
   return;
}
```

Al igual que con JButton, el evento que ocurre cuando el usuario selecciona un botón de radio es: ActionPerformed.

JSlider

Barra deslizante. Componente que permite seleccionar un valor en un intervalo deslizando un botón



Para obtener el valor seleccionado tenemos el método getValue() y el evento para detectar cambios en un Slider sería:

```
jSlider1.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
          // Aquí el código
    }
});
```

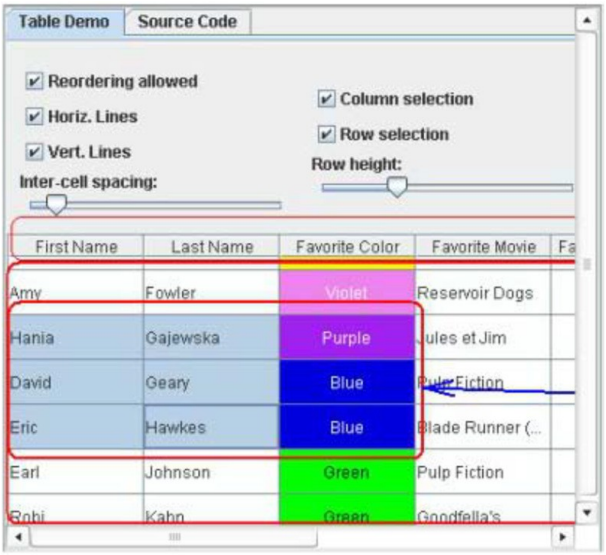
JSpinner

Un cuadro de entrada de una línea que permite seleccionar un número (o el valor de un objeto) de una secuencia ordenada. Normalmente proporciona un par de botones de flecha, para pasar al siguiente o anterior número, (o al siguiente o anterior valor de la secuencia).

JSpiner permite seleccionar un valor de una secuencia ordenada de valores posibles. Normalmente se usa con datos numéricos y las flechas permiten seleccionar el anterior (abajo) o siguiente número o valor de la secuencia. También puede escribirse el valor directamente.

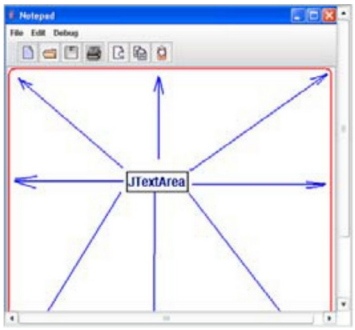
JTable

Una tabla bidimensional para presentar datos



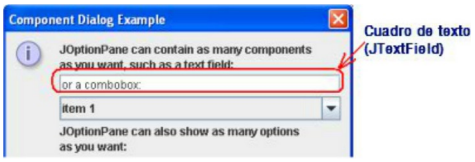
JTextArea

Crea un área de texto de dos dimensiones para texto plano



JtextField

Crea un cuadro de texto de una dimensión.

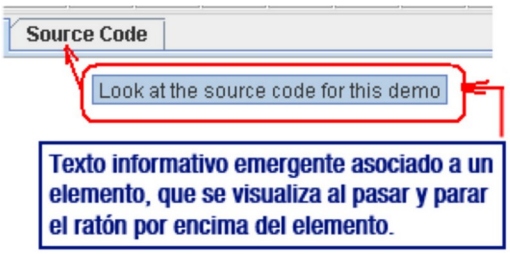


Los dos métodos que más nos interesan son el que obtienen el texto que hay escrito en el componente: getText() y el que escribe texto en el componente: setText().

El valor devuelto por getText(); es de tipo String por lo que, si queremos hacer operaciones numéricas el mismo, debemos convertirlo con los métodos estáticos que ya conocemos, como Integer.parseInt().

JToolTip

Una especie de etiqueta emergente, que aparece para visualizar la utilidad de un componente cuando el cursor del ratón pasa por encima de él deteniéndose brevemente, sin necesidad de pulsar nada.



JTree

Visualiza un conjunto de datos jerárquicos en forma de árbol

4.5 Métodos comunes

Existen una serie de métodos comunes a los elementos que acabamos de ver, de uso frecuente:

- setBounds (x,y,w,h); Este es común a todos los componentes e indica la posición y el tamaño del "rectángulo" que forma la apariencia del componente. Los primeros dos parámetros son las posiciones X e Y de la esquina superior izquierda (es la posición respecto a su contenedor). El tercer parámetro es el ancho del elemento y el cuarto el alto del elemento.
- setLocation (x,y); y setSize (w,h); representan lo mismo que el anterior, por separado.
- setResizeable (bool); Indica si se permite cambiar el tamaño del componente o no.
- setVisible (bool);
- setEnable (bool); si acepta eventos o no (aparecerá en gris)
- setText(String n): para establecer el texto que aparece en el elemento (etiqueta, botón, caja de texto, etc.)
- setName(String n); para asingnarle un nombre al elemento.
- getName(); para obtener el nombre del objeto.

4.6 Ventanas de Diálogo

Las ventanas de diálogo son ventanas sencillas, que se muestran en forma de “pop-up”, que muestran información y pueden pedir información al usuario. Otra característica fundamental es que son modales, es decir, paran la ejecución de nuestro programa hasta que el usuario cierre el diálogo seleccionando alguna de las opciones.

Las ventanas de diálogo se gestionan en Swing con la clase JOptionPane que contiene los métodos básicos para cada tipo de diálogo. Todos los métodos que vamos a ver tienen dos parámetros comunes a todos ellos:

- parentComponent: Apuntador al padre (ventana sobre la que aparecerá el diálogo) o bien null. En muchos casos podrá ser this, la ventana donde está definido.
- message: El mensaje a mostrar, habitualmente un String, aunque vale cualquier Object cuyo método toString() esté definido.

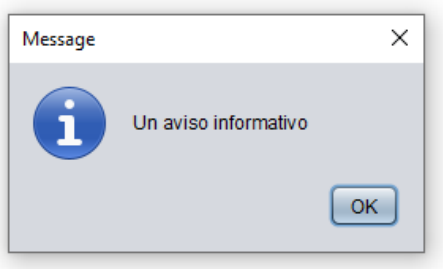
Estos son los métodos y los diálogos que generan:

JoptionPane.showMessageDialog()

Este es el diálogo más sencillo de todos, sólo muestra una ventana de aviso al usuario. La ejecución se detiene hasta que el usuario cierra la ventana. El método está sobrecargado, con más o menos parámetros, en función de si aceptamos las opciones por defecto o deseamos asignarle un título y un icono. Ejemplo:

```
JOptionPane.showMessageDialog(this, "Un aviso informativo");
```

Mostrando:



No devolviendo ningún valor y continuando la ejecución del programa una vez se pulse Aceptar.

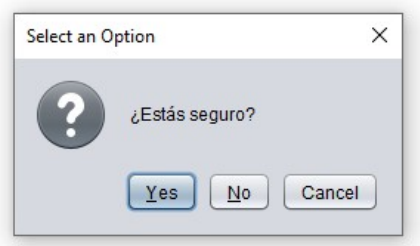
JOptionPane.showConfirmDialog()

Este método muestra una ventana pidiendo una confirmación al usuario, estilo "¿Estás seguro?" y da al usuario opción de aceptar o cancelar esa operación. El método devuelve un entero indicando la respuesta del usuario. Los valores de ese entero pueden ser alguna de las constantes definidas en JOptionPane: YES_OPTION, NO_OPTION, CANCEL_OPTION, OK_OPTION, CLOSED_OPTION.

El siguiente ejemplo de código

```
int respuesta = JOptionPane.showConfirmDialog(this, "¿Estás seguro?");
if (respuesta == JOptionPane.OK_OPTION)
   System.out.println("El usuario confirma la operación");
else System.out.println("El usuario cancela la operación");
```

muestra la siguiente imagen:



Podríamos configurar las opciones de respuesta con nuestros propios valores, utilizando el método showOptionDialog que veremos más adelante.

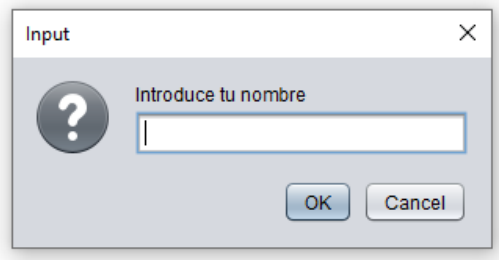
JOptionPane.showInputDialog()

A diferencia del anterior, este diálogo permite una respuesta del usuario. También está sobrecargado, admitiendo más o menos parámetros, según queramos aceptar o no las opciones por defecto. Los parámetros y sus significados son muy similares a los del método showOptionDialog(), pero hay una diferencia.

Si usamos los métodos que no tienen array de opciones, la ventana mostrará una caja de texto para que el usuario escriba la opción que desee (un texto libre). Si usamos un método que tenga un array de opciones, entonces aparecerá en la ventana un JComboBox en vez de una caja de texto, donde estarán las opciones que hemos pasado. Vemos las dos posibilidades en estos dos ejemplos:

```
String txt = JOptionPane.showInputDialog(this, "Introduce tu nombre");
System.out.println("El usuario ha escrito "+ txt);
```

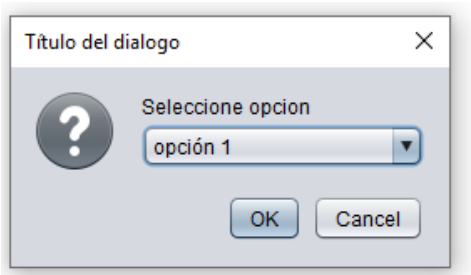
y la imagen que obtenemos con este código:



En este segundo ejemplo, incluimos JComboBox con las opciones:

```
Object txt = JOptionPane.showInputDialog(null, "Seleccione opcion",
   "Título del dialogo", JOptionPane.QUESTION_MESSAGE, null,
   new String[] { "opción 1", "opción 2", "opción 3" }, "opcion 1");
System.out.println("El usuario ha elegido "+ txt);
```

y esta es la imagen que se obtiene.



En ambos casos, si se pulsa Cancel o la X de cierre de ventana, el valor obtenido es null.

JOptionPane.showOptionDialog()

Este método muestra la ventana más configurable de todas, en ella debemos definir todos los botones que lleva. De hecho, las demás ventanas disponibles con JOptionPane se construyen a partir de esta. Por ello, al método debemos pasarle muchos parámetros:

- parentComponent: Apuntador al padre (ventana sobre la que aparecerá el diálogo) o bien null (ventana actual).
- message: El mensaje a mostrar, habitualmente un String, aunque vale cualquier Object cuyo método toString() esté definido.
- title: El título para la ventana.
- optionType: Un entero indicando qué botones tendrá el diálogo. Los posibles valores son las constantes
- definidas en JOptionPane: DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION, o OK_CANCEL_OPTION.

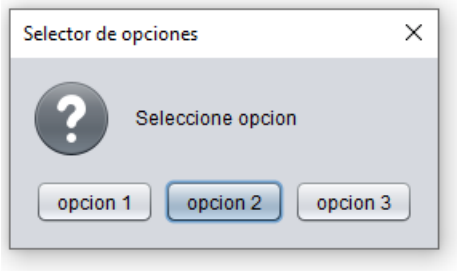
- messageType: Un entero para indicar qué tipo de mensaje estamos mostrando. Este tipo servirá para que se determine qué icono mostrar. Los posibles valores son constantes definidas en JOptionPane: ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, o PLAIN_MESSAGE
- icon: Un icono para mostrar. Si ponemos null, saldrá el icono adecuado según el parámetro messageType.
- options: Un array de objects que determinan las posibles opciones. Si los objetos son componentes visuales, aparecerán tal cual como opciones. Si son String, el JOptionPane pondrá tantos botones como String. Si son cualquier otra cosa, se les tratará como String llamando al método toString(). Si se pasa null, saldrán los botones por defecto que se hayan indicado en optionType.
- initialValue: Selección por defecto. Debe ser uno de los Object que hayamos pasado en el parámetro options. Se puede pasar null.

La llamada a JOptionPane.showOptionDialog() devuelve un entero que representa la opción que ha seleccionado el usuario. La primera de las opciones del array es la posición cero. Si se cierra la ventana con la cruz de la esquina superior derecha, el método devolverá -1.

Aquí un ejemplo de cómo llamar a este método:

```
int seleccion = JOptionPane.showOptionDialog(
null, "Seleccione opcion", "Selector de opciones", //ventana, mensaje, título
JoptionPane.YES_NO_CANCEL_OPTION, JoptionPane.QUESTION_MESSAGE, null,
// icono: null para icono por defecto
new Object[]{"opcion 1","opcion 2","opcion 3" }, // opciones
//null para YES, NO y CANCEL
"opcion 2"
//opción con el foco
);
```

y la ventana que se obtiene:



JDialog

Por último, comentar que disponemos de la clase JDialog, que es una clase base para hacer ventanas de diálogo totalmente personalizadas. Son un elemento más que se puede añadir a nuestro Jframe y podemos añadirle cualquier componente como cajas de texto, botones de opción, etc. Exactamente igual que hacemos con una ventana o con un panel.

A nivel de código haremos lo siguiente:

- En el constructor del JDialog añadiremos:

```
setLocationRelativeTo(null);
this.setVisible(true);
```

- El JDialog deberá tener un botón de cierre con el código:

```
this.setVisible(false);
```

- Definiremos las variables que queremos que el usuario asigne valor como atributos del JDialog y al cerrar la ventana tomarán valores a partir de los componentes incluidos en el diálogo. También podemos hacer validaciones y deshabilitar ese botón mientras no se cumplan ciertas condiciones.
- Para finalizar, desde el JFrame de nuestra aplicación llamaremos al diálogo. Por ejemplo, si le habíamos llamado MyDialog:

```
MyDialogCheckIn myD = new MyDialog(new javax.swing.JFrame(), true);
```

- Al cerrar el diálogo tendremos disponibles los valores de los atributos para tratarlos en nuestra aplicación: myD.atributo1, myD.atributo2, etc.

4.7 Menús

JMenuBar

Una barra de menú, que aparece generalmente situada debajo de la barra de título de la ventana

JMenu

Un menú de lista desplegable, que incluye JMenuItems y que se visualiza dentro de la barra de menú (JmenuBar)

JMenuItem

Cada una de las opciones de un menú concreto

5 Gestión de Eventos

Ya hemos visto en los apartados anteriores eventos típicos como seleccionar un botón o detectar los cambios en un Slider. A continuación, vamos a profundizar un poco más en la gestión de eventos.

5.1 Eventos comunes a varios componentes

El editor visual crea código de eventos para cada componente por separado. Un enfoque diferente sería hacer métodos generales comunes a varios elementos y mediante getSource obtener el elemento concreto que lo ha producido.

Cuando tenemos Arrays de elementos, esta es la forma habitual de trabajar (por ejemplo, en un tablero, los dígitos de una calculadora, etc). No tendría sentido duplicar de varios elementos que realizan la misma función (en la calculadora, la tarea a realizar si pulsas el 1 o el 2 o 3... es la misma, solo necesitamos identificar el botón que se ha pulsado.

Analiza este ejemplo, que dibuja un tablero de damas y le asigna a cada casilla (que en realidad es un JButton) un nombre compuesto por la fila y columna en la que está ubicado.

```
import javax.swing.*;
import java.awt.Color;
public class damas extends javax.swing.JFrame {
    private JButton tablero[][];
    public damas() {
        boolean blanco = true;
        tablero = new JButton[8][8];
        for (int f = 0; f < 8; f++) {
            for (int c = 0; c < 8; c++) {
                tablero[f][c] = new JButton();
```

```
                tablero[f][c].addActionListener(new java.awt.event.ActionListener() {
                    public void actionPerformed(java.awt.event.ActionEvent evt) {
                        tareaSipulsa(evt);
                    }
                });
                tablero[f][c].setName(Integer.toString(f) + Integer.toString(c));
                add(tablero[f][c]);
                tablero[f][c].setBounds(c * 30, f * 30, 34, 34);
                if (blanco) {
                    tablero[f][c].setBackground(Color.white);
                    blanco = false;
                } else {
                    tablero[f][c].setBackground(Color.black);
                    blanco = true;
                }
            }
            if (blanco) blanco = false;
            else blanco = true;
        }
    }
    private void tareaSipulsa(java.awt.event.ActionEvent evt) {
            evt.getSource();
            //(JButton) evt.getSource() sería el botón pulsado
        }
        ...
}
```

En este ejemplo, hemos situado los botones “a mano” en la pantalla, mediante el método setBounds(). Más adelante, cuando veamos los Layouts, veremos que hay formas más eficientes y rápidas de colocar los objetos con la disposición que queramos. El caso de los tableros, la disposición GridLayout o GridBagLayout será la más aconsejable.

5.2 Lista de Eventos

NOMBRE LISTENER	DESCRIPCIÓN	MÉTODOS	EVENTOS
ActionListener	Se produce al hacer click en un componente, también si se pulsa Enter teniendo el foco en el componente.	public void actionPerformed (ActionEvent e)	**JButton**: click o pulsar Enter con el foco activado en él. **JList**: doble click en un elemento de la lista. **JMenuItem**: selecciona una opción del menú. **JTextField**: al pulsar Enter con el foco activado.
KeyListener	Se produce al pulsar una tecla. según el método cambiara la forma de pulsar la tecla.	public void keyTyped(KeyEvent e)	Cuando pulsamos una tecla, según el Listener: keyTyped: al pulsar y soltar la tecla.
		public void keyPressed (KeyEvent e)	keyPressed: al pulsar la tecla.
		public void keyReleased (KeyEvent e)	keyReleased: al soltar la tecla.

FocusListener	Se produce cuando un componente gana o pierde el foco, es decir, que esta seleccionado.	public void focusGained (FocusEvent e)	Recibir o perder el foco.
		public void focusLost (FocusEvent e)	
MouseListener	Se produce cuando realizamos una acción con el ratón.	public void mouseClicked (MouseEvent e)	Según el Listener: mouseCliked: pinchar y soltar.
		public void mouseEntered (MouseEvent e)	mouseEntered: entrar en un componente con el puntero.
		public void mouseExited (MouseEvent e)	mouseExited: salir de un componente con el puntero
		public void mousePressed (MouseEvent e)	mousePressed: presionar el botón.
		public void mouseReleased (MouseEvent e)	mouseReleased: soltar el botón.
		public void mouseDragged (MouseEvent e)	Según el Listener: mouseDragged: click y arrastrar un componente.
		public void mouseMoved (MouseEvent e)	mouseMoved: al mover el puntero sobre un elemento
TextListener	Cambios en texto	textValueChanged();	Modificar el texto
ItemListener	Cambios en checkbox	ItemStateChanged()	Clicar en una checkbox

6 Paneles y Layout Managers

Los paneles son contenedores para otros componentes y los usamos por diversos motivos:

- Para organizar los componentes. Si nuestro JFrame contiene muchos elementos, podemos dividirlos en distintas áreas, cada una con su panel.
- Para aplicaciones multiventana. Tendremos un solo JFrame con varios paneles (que normalmente ocuparán todo el JFrame) y según las distintas opciones que seleccione el usuario se mostrará uno y otro panel.
- Agrupar acciones sobre varios componentes. Si queremos hacer un tratamiento especial sobre un conjunto de elementos (por ejemplo, moverlos, hacerlos invisibles, etc.) si están en un panel, podremos hacerlo sobre el panel en vez de cada elemento uno a uno.

- Controlar la disposición de los componentes gráficos: en fila, haciendo un tablero, etc.
- Capas. Si nuestra aplicación muestra distintas capas que se superponen, los paneles pueden hacer esta función.

Ya comentamos que todo JFrame tiene un panel por defecto, pero podemos crear nosotros distintos paneles mediante el componente o la clase JPanel.

Una vez creados, trabajamos con ellos como un JFrame. Podemos mostrarlos u ocultarlos con el método setVisible (boolean).

Para probarlo, crea un JFrame con:

```
import javax.swing.JPanel;
```

y añádele dos atributos de tipo JPanel:

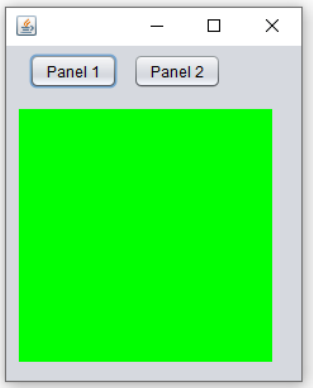
```
private JPanel jPanel1;
private JPanel jPanel2;
```

Luego instanciamos los paneles y los añadimos al JFrame.

```
jPanel1 = new JPanel(null);
jPanel1.setBounds(10, 50, 200, 200);
jPanel1.setBackground(java.awt.Color.GREEN);
this.add(jPanel1);
jPanel1.setVisible(true);
jPanel2 = new JPanel(null);
jPanel2.setBounds(10, 50, 200, 200);
jPanel2.setBackground(java.awt.Color.RED);
jPanel2.setVisible(false);
this.add(jPanel2);
```

Podríamos añadir al JFrame dos botones que hiciesen que estuviese visible uno u otro panel:

```
private void jButton1ActionPerformed
(java.awt.event.ActionEvent evt) {
        jPanel1.setVisible(true);
        jPanel2.setVisible(false);
    }
private void jButton2ActionPerformed
(java.awt.event.ActionEvent evt) {
        jPanel1.setVisible(false);
        jPanel2.setVisible(true);
    }
```



La forma de trabajar con los JPanel es como con los JFrame, es decir, añadiendo los componentes al mismo mediante el método add().

6.1 Layout

Los paneles nos ofrecen distintas formas de organizar, de manera automática, la posición y tamaño de los componentes dentro de los contendedores, mediante lo que se conoce como Layout, Managers o gestores de aspecto.

FlowLayout

Es el más simple y el que se utiliza por defecto en todos los paneles si no se indica el uso de alguno de los otros. Los componentes añadidos a un panel con FlowLayout se encadenan en forma de lista. La cadena es horizontal, de izquierda a derecha, y se puede seleccionar el espaciado entre cada componente.

Si el contenedor se cambia de tamaño en tiempo de ejecución, las posiciones de los componentes se ajustarán automáticamente, para colocar el máximo número posible de componentes en la primera línea.

Los componentes se alinean según se indique en el constructor. Si no se indica nada, se considera que los componentes que pueden estar en una misma línea estarán centrados, pero también se puede indicar que se alineen a izquierda o derecha en el contenedor.

BorderLayout

Esta composición (de borde) proporciona un esquema más complejo de colocación de los componentes en un panel. Es el layout o composición que utilizan por defecto JFrame y Jdialog. La composición utiliza cinco zonas para colocar los componentes sobre ellas:

- Norte: parte superior del panel (NORTH o PAGE_START)
- Sur: parte inferior del panel (SOUTH o PAGE_END)
- Este: parte derecha del panel (EAST o LINE_END)
- Oeste: parte izquierda del panel (WEST o LINE_START)
- Centro: parte central del panel, una vez que se hayan rellenado las cuatro partes (CENTER)

Este controlador de posicionamiento resuelve los problemas de cambio de plataforma de ejecución de la aplicación, pero limita el número de componentes que pueden ser colocados en contenedor a cinco; aunque, si se va a construir un interfaz gráfico complejo, algunos de estos cinco componentes pueden contenedores, con lo cual el número de componentes puede verse ampliado.

En los cuatro lados, los componentes se colocan y redimensionan de acuerdo a sus tamaños preferidos y a los valores de separación que se hayan fijado al contenedor.

Es muy importante indicar:

- Tamaño prefijado: setPreferredSize(new Dimension (WIDTH,HEIGHT));
- Tamaño máximo: setMaximumSize(new Dimension(WIDTH,HEIGHT));
- Tamaño mínimo: setMinimumSize(new Dimension(WIDTH,HEIGHT));

ya que un botón o panel puede ser redimensionado a proporciones cualesquiera; sin embargo, el diseñador puede fijar un tamaño preferido para obtener una mejor visualización de cada componente.

CardLayout

Este es el tipo de composición que se utiliza cuando se necesita una zona de la ventana que permita colocar distintos componentes en esa misma zona. Este layout suele ir asociado con botones de selección, de tal modo que cada selección determina el panel (grupo de componentes) que se presentarán.

GridLayout

Esta composición proporciona gran flexibilidad para situar componentes. El controlador de posicionamiento se crea con un determinado número de filas y columnas y los componentes van dentro de las celdas de la tabla así definida.

Si el contenedor es alterado en su tamaño en tiempo de ejecución, el sistema intentará mantener el mismo número de filas y columnas dentro de los márgenes de separación que se hayan indicado. En este caso, estos márgenes tienen prioridad sobre el tamaño mínimo que se haya indicado para los componentes, por lo que puede llegar a conseguirse que sean de un tamaño tan pequeño que sus etiquetas sean ilegibles.

GridBagLayout

Es igual que la composición de GridLayout, con la diferencia que los componentes no necesitan tener el mismo tamaño. Es quizá el controlador de posicionamiento más sofisticado de los que actualmente soporta Swing.

BoxLayout

El controlador de posicionamiento BoxLayout es uno de los dos que incorpora Java a través de Swing y permite colocar los componentes a lo largo del eje X o del eje Y. También posibilita que los componentes ocupen diferente espacio a lo largo del eje principal.

En un controlador BoxLayout sobre el eje Y, los componentes se posicionan de arriba hacia abajo en el orden en que se han añadido. Al contrario, en en el caso del GridLayout, aquí se permite que los

componentes sean de diferente tamaño a lo largo del eje Y, que es el eje principal del controlador de posicionamiento, en este caso.

En el eje que no es principal, BoxLayout intenta que todos los componentes sean tan anchos como el más ancho, o tan alto como el más alto, dependiendo de cuál sea el eje principal. Si un componente no puede incrementar su tamaño, el BoxLayout mira las propiedades de alineamiento en X e Y para determinar dónde colocarlo.

OverlayLayout

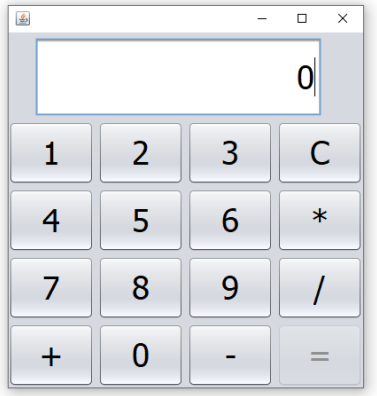
El controlador de posicionamiento OverlayLayout, es el otro que se incopora a Java con Swing, y es un poco diferente a todos los demás. Se dimensiona para contener el más grande de los componentes y superpone cada componente sobre los otros.

La clase OverlayLayout no tiene un constructor por defecto, así que hay que crearlo dinámicamente en tiempo de ejecución.

Por último, comentar que es posible construir nuestros propios layouts (CustomLayout).

6.2 Ejemplo: Calculadora con GridLayout

Vamos a hacer paso a paso una sencilla calculadora, empleando diferentes Layouts. La apariencia final que buscamos es como la de la siguiente figura.



En ella tendremos un JPanel para la parte superior y otro JPanel para la botonera. Este segundo panel será tipo GridLayout de 4 filas x 4 columnas. Al añadir al panel los botones se irán colocando uno a continuación de otro ocupando las 16 posiciones.

1.- Sobre el paquete de nuestro proyecto, botón derecho > New Frame. A continuación, desde la ventana navegador, botón derecho sobre ella > Set Layout > Border Layout.

2. – Arrastramos un JPanel desde la paleta a la parte superior del JFrame, y lo soltamos cuando la línea de puntos amarilla nos muestre la zona superior del frame. Esto indica que estamos situando el panel en el

área North. Desde la ventana Navigator, botón derecho sobre este panel: lo llamamos Cabecera cambiándole el nombre de la variable y decimos que tiene BorderLayout. Luego veremos que contendrá la caja de texto en la que se muestran operandos y resultados.

3.- Arrastramos otro JPanel desde la paleta a la parte central del frame, y le llamaremos Botonera. Desde la ventana Navigator, botón derecho sobre este panel: Set Layout > GridLayout. Luego le asignaremos todos los botones.

En las propiedades de ambos paneles podemos fijar su PreferedSize para establecer su tamaño, 400 x 100 para el primero y 400 x 300 para el segundo.

4.- Arrastramos al panel de cabecera un JTextField y cambiamos en la paleta su formato como tamaño de letra, alineación, etc.

5.- Modificamos el código del JFrame para añadir los 16 botones que forman la calculadora al panel Botonera e incorporamos las variables globales para hacer los cálculos:

```
public class Calculadora extends javax.swing.JFrame {
        long num1
        long num2
        char oper
        JButton[] = 0; = 0; = '=';
        tablero;
        //definimos el array con los 16 botones
        public Calculadora() {
                initComponents();
                setLocationRelativeTo(null);
                //ventana centrada en pantalla
                tablero = new JButton[16];
                //definimos los 16 botones
                for (int i = 0; i < 16; i++) {
                    tablero[i] = new JButton();
                    tablero[i].addActionListener(new java.awt.event.ActionListener() {
                        public void actionPerformed(java.awt.event.ActionEvent evt) {
                            FActionPerformed(evt);
                        }
                    });
                    tablero[i].setName("jButton" + Integer.toString(i));
                    tablero[i].setFont(new java.awt.Font("Tahoma", 0, 36));
                    if (i < 10) {
                        tablero[i].setText(Integer.toString(i));
                    } else {
                        switch (i) {
                            case 10: tablero[i].setText("+"); break;
                            case 11: tablero[i].setText("-"); break;
                            case 12: tablero[i].setText("C"); break;
                            case 13: tablero[i].setText("*"); break;
                            case 14: tablero[i].setText("/"); break;
                            case 15: tablero[i].setText("="); break;
                        }
                    }
                }
                //añadimos los botones al Grid Panel en el orden que queremos que los muestre
                Botonera.add(tablero[1]); Botonera.add(tablero[2]); Botonera.add(tablero[3]);
                Botonera.add(tablero[12]); Botonera.add(tablero[4]); Botonera.add(tablero[5]);
```

```
                Botonera.add(tablero[6]); Botonera.add(tablero[13]); Botonera.add(tablero[7]);
                Botonera.add(tablero[8]); Botonera.add(tablero[9]); Botonera.add(tablero[14]);
                Botonera.add(tablero[10]); Botonera.add(tablero[0]); Botonera.add(tablero[11]);
                Botonera.add(tablero[15]);
                tablero[15].setEnabled(false); //boton "=" deshabilitado
```

6.- Para finalizar, incorporamos el código que se ejecuta al pulsar un botón, esto es, el método FActionPerformed (evt). Será un código común para los 10 botones que representan dígitos.

```
private void FActionPerformed(java.awt.event.ActionEvent evt) {
    String nomBoton = ((JButton) evt.getSource()).getName();
    int numBoton = Integer.parseInt(nomBoton.substring(7, nomBoton.length()));
    if (numBoton <= 9) {
        //si es un botón numérico
        if (jPantalla.getText().equals("0"))
            jPantalla.setText(((Integer) numBoton).toString());
        else
            jPantalla.setText(jPantalla.getText() +
                ((Integer) numBoton).toString());
    } else {
        switch (numBoton) {
            case 10: //boton +
                num1 = Long.parseLong(jPantalla.getText());
                oper = '+';
                jPantalla.setText("0");
                tablero[15].setEnabled(true);
                break;
            case 11: //boton -
                num1 = Long.parseLong(jPantalla.getText());
                oper = '-';
                jPantalla.setText("0");
                tablero[15].setEnabled(true);
                break;
            case 12: //letra "C" borra el último digito introducido
                int longit = jPantalla.getText().length();
                if (longit > 1)
                    jPantalla.setText(
                        jPantalla.getText().substring(0, longit - 1));
                else {
                    jPantalla.setText("0");
                }
                break;
            case 13: // boton *
                num1 = Long.parseLong(jPantalla.getText());
                oper = '*';
                jPantalla.setText("0");
                tablero[15].setEnabled(true);
                break;
            case 14: // boton /
                num1 = Long.parseLong(jPantalla.getText());
                oper = '/';
                jPantalla.setText("0");
                tablero[15].setEnabled(true);
                break;
            case 15: // boton =
                long resul = 0;
                num2 = Long.parseLong(jPantalla.getText());
                switch (oper) {
                    case '+': resul = num1 + num2; break;
```

```
                    case '-': resul = num1 - num2; break;
                    case '*': resul = num1 * num2; break;
                    case '/':
                        if (num2 != 0) resul = num1 / num2;
                        else resul = 0;
                        break;
                }
                jPantalla.setText(Long.toString(resul));
                tablero[15].setEnabled(false);
                break;
        }
    }
}
```

7 Gráficos y Animaciones

La animación gráfica queda fuera del ámbito de este curso ya que es un tema muy especializado y existen librerías específicas para optimizar tanto el desarrollo como la ejecución. Sin embargo, vamos a ver utilidades sencillas que pueden resultarnos útiles en nuestras aplicaciones, para hacerlas más vistosas o programar juegos sencillos.

7.1 Imágenes

La forma más sencilla de incluir imágenes en nuestras aplicaciones es añadiendo una etiqueta Jlabel, eliminando su texto y añadiéndole la imagen mediante el método setIcon (en Netbeans: propiedad "Icon"). Podemos incluir imágenes de diferentes tipos incluidos gif animados.

7.2 Swing.Timer

En nuestras aplicaciones con interfaz visual es muy frecuente que tengamos que ejecutar algo periódicamente, por ejemplo, un contador, un reloj, o el movimiento de un objeto por la ventana, etc. Para ello tenemos varias opciones: uso de hilos (Threads), la clase general Timer (del paquete util) y la clase Timer (del paquete Swing). La primera es la más completa pero también la más compleja.

La segunda y la tercera opción son más sencillas, y la ventaja principal de la última es que el código se programa en la propia clase del contenedor, por lo que tenemos disponibles los objetos que lo componen sin necesidad de tener que pasárselos por parámetro, que sería lo que habría que hacer en caso de usar util.Timer.

```
myTimer = new Timer (1000, new java.awt.event.ActionListener () {
   public void actionPerformed(java.awt.event.ActionEvent e)
         {
               //Aquí la tarea a repetir, por ejemplo:
               // jTextField1.setText(LocalDateTime.now().toString());
```

```
        }
});
```

Y luego podríamos arrancar o parar la tarea con:

```
myTimer.start();
```

y pararla con

```
myTimer.stop();
```

7.3 Moviendo elementos

Un aspecto fundamental en las aplicaciones gráficas es el movimiento de los elementos que aparecen en pantalla. Para ello, modificaremos su posición mediante el método setLocation(x,y) representando 'x' e 'y' las coordenadas de la esquina superior izquierda del elemento.

Estas coordenadas no son absolutas respecto a la aplicación, sino que son relativas al contenedor en el que esté ubicado el elemento, ya sea el JFrame o un JPanel concreto.

Si lo que necesitamos es un desplazamiento, tendremos que saber las coordenadas actuales para incrementarlas o decrementarlas. Esa información la obtenemos con los métodos getX() y getY() y finalmente necesitaremos saber los límites por los que nos podemos desplazar. Lo haremos con los métodos getHeight() y getWidth() pero del contenedor del elemento, no del propio elemento (en muchos casos será this ya que programamos eventos en el contenedor).

El siguiente ejemplo movería la etiqueta en diagonal hacia abajo.

```
jLabel1.setLocation(jLabel1.getX()+1, jLabel1.getY()+1);
```

Este movimiento puede ser automático, con Swing.Timer o provocado por acciones del usuario mediante los eventos vistos previamente.

7.4 Dibujando figuras

Para dibujar figuras el proceso que seguiremos será el siguiente:

1. Definir un panel (una nueva clase que extiende JPanel).
2. Sobrescribir el método paintComponent () del panel
3. Añadir el panel al JFrame

4. Cada vez que haya un cambio, llamar a repaint() del panel, el cual llama internamente a paintComponent ()

En la clase Graphics disponemos de métodos para dibujar figuras básicas. Por ejemplo:

```
@Override
public void paintComponent(Graphics g) {
   super.paintComponent(g); // siempre es la primera instrucción
   g.drawString("Ejemplo de texto",30,20); // texto y posición
   g.setColor(Color.RED);
   // color de primer plano
   g.fillRect(40,50,60,70); // rectángulo sólido
   g.setColor(Color.BLUE);
   // cambia el color de primer plano
   g.drawRect(80,90,100,120);
   // dibuja un rectángulo sin relleno
   g.drawLine(0, 0, 70, 70); // línea de esquina sup. izq. a inf. dcha
}
```

Ver esta serie de videos sobre cómo hacer el juego “Pong” con Swing en la que se ven los conceptos vistos previamente: https://www.youtube.com/watch?v=fnJQLBPemcI